Week 4 HW_Python Review

January 28, 2021

1 Week 4 Individual Coding Assignment

Author: Kimberly Venegas

Kim and Brian's research question is: how accessible are public parks and bike lanes in low-income communities of color in Los Angeles?

I will create an isochrone map for View Park-Windsor Hills, Los Angeles, CA. I chose that neighborhood because it has a large percentage of Black people and I want to use this data to later analyze how long it takes residents to walk to their nearest park.

1.1 Importing the libraries

[1]:
```
import geopandas as gpd

import matplotlib.pyplot as plt

import networkx as nx

import osmnx as ox

import contextily as ctx

from shapely.geometry import Point, LineString, Polygon
from descartes import PolygonPatch
```

1.2 Importing and converting the dataset

I will get the network data from OpenStreetMap using OSMnx and ask for a walking network type.

[2]:
```
place = 'View Park-Windsor Hills, Los Angeles, CA, USA'
network_type = 'walk'
trip_times = [5, 10, 15, 20]
meters_per_minute = 75
```

I will download the street network and plot it.

[3]:
```
G = ox.graph_from_place(place, network_type=network_type)
fig, ax = ox.plot_graph(G)
```

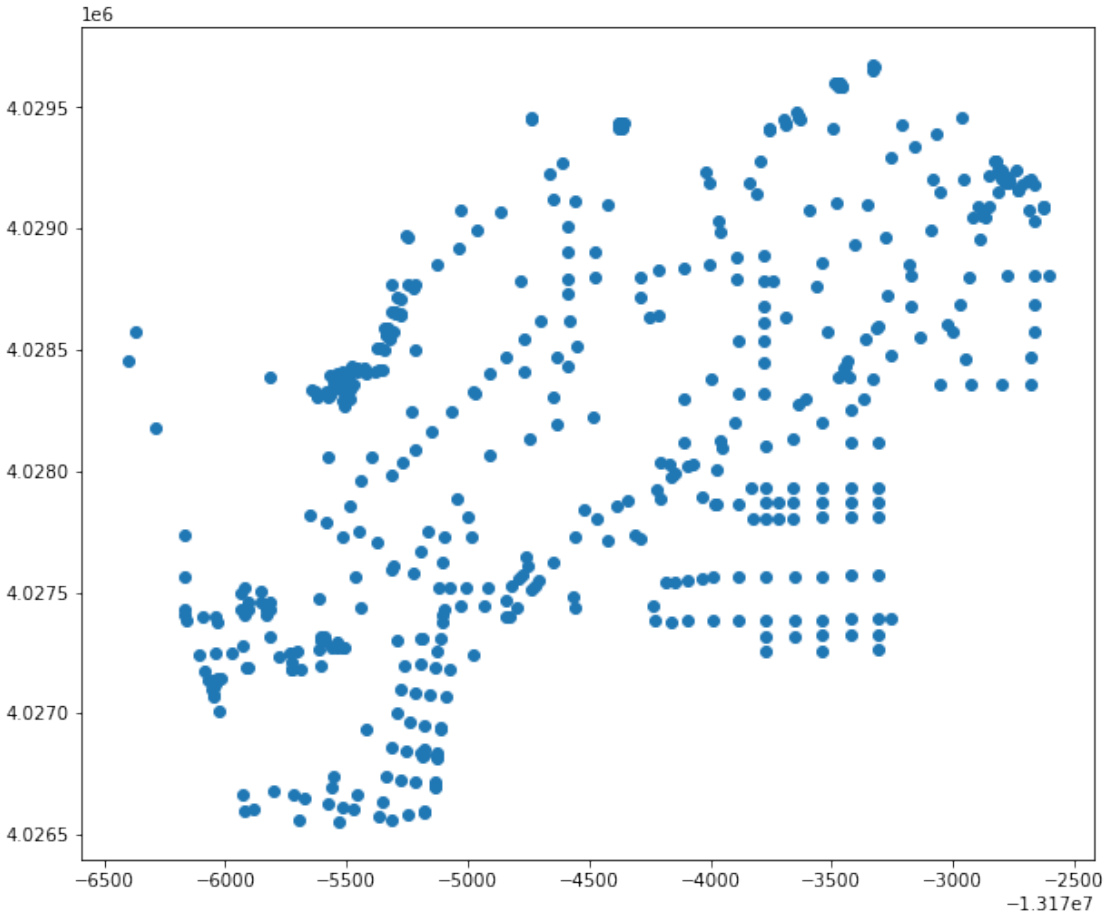
I'm going to project the data to Web Mercator and create separate geodataframes for edges and nodes.

[4]:
```
G = ox.project_graph(G, to_crs='epsg:3857')
gdf_nodes, gdf_edges = ox.graph_to_gdfs(G)
```

Let's see what the nodes geodataframe looks like.

[5]:
```
gdf_nodes.plot(figsize=(10,10))
```

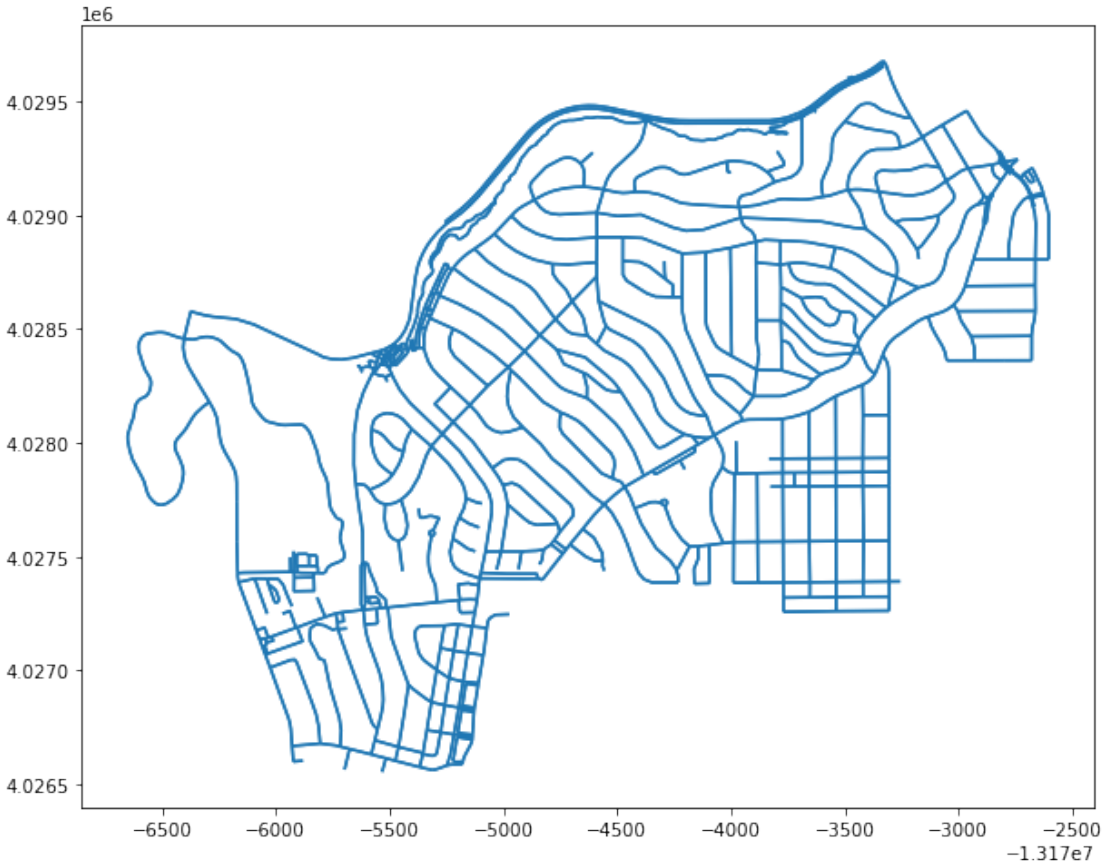
[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62829e2b80>
```



Let's see what the edges geodataframe looks like.

[6]:
```
gdf_edges.plot(figsize=(10,10))
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6283dc12e0>
```



1.3 Calculating basic street network measures

I am going to get the bounding box coordinates.

[7]:
```
minx, miny, maxx, maxy = gdf_nodes.geometry.total_bounds
print(minx)
print(miny)
print(maxx)
print(maxy)
```

```
-13176402.538449002
4026553.118636329
-13172603.64950619
4029676.01900891
```

Now I will calculate the centroid.

[8]:
```
centroid_x = (maxx-minx)/2 + minx
centroid_y = (maxy-miny)/2 + miny
print(centroid_x)
print(centroid_y)
```

```
-13174503.093977597
4028114.5688226195
```

Now I am going to get rid of the nearest node by using nearest_node command.

[9]:
```
center_node = ox.get_nearest_node(G,
                                  (centroid_y,centroid_x),
                                  method = 'euclidean')
print('The id for the nearest node is ' + str(center_node))
```

```
The id for the nearest node is 122975580
```

I want to see the record.

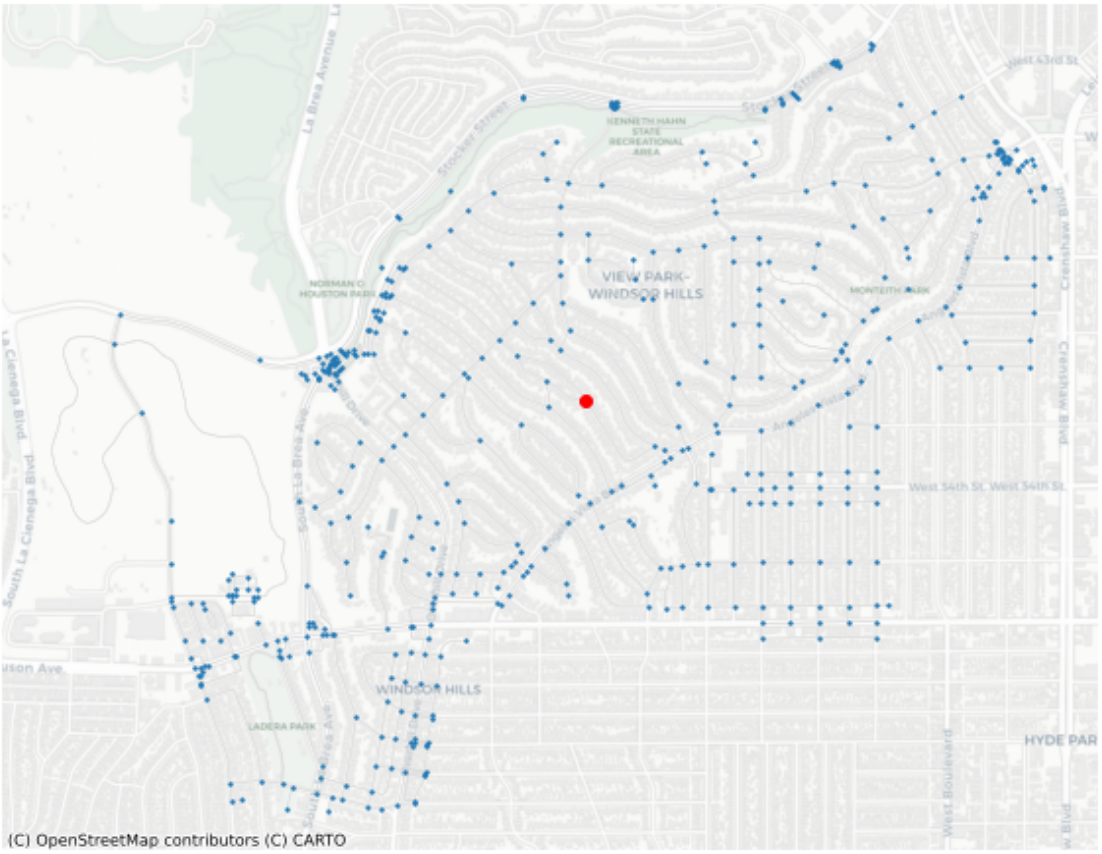
[10]:
```
gdf_nodes.loc[[center_node]]
```

[10]:

	y	x	osmid	highway	lon	lat	geometry
122975580	4.028222e+06	-1.317448e+07	122975580	NaN	-118.348387	33.995681	POINT (-13174482.144 4028222.118)

Let's see a map of the nodes.

[11]:
```
fig, ax = plt.subplots(figsize=(10,10))

gdf_edges.plot(ax=ax,
               linewidth=0.5,
               edgecolor='gainsboro',
               zorder=10)

gdf_nodes.plot(ax=ax,
               markersize=2,
               zorder=20)

gdf_nodes.loc[[center_node]].plot(ax=ax,
                                  color='r',
                                  zorder=30)

ax.axis('off')

ctx.add_basemap(ax,source=ctx.providers.CartoDB.Positron)
```



1.4 Creating an isochrone map

I am going to use the function below to create the isochrones.

[12]:
```
def make_iso_polys(G, edge_buff=25, node_buff=50, infill=False):
    isochrone_polys = []
    for trip_time in sorted(trip_times, reverse=True):
        subgraph = nx.ego_graph(G, center_node, radius=trip_time,␣
↪distance='time')

        node_points = [Point((data['x'], data['y'])) for node, data in subgraph.
↪nodes(data=True)]
        nodes_gdf = gpd.GeoDataFrame({'id': list(subgraph.nodes)},␣
↪geometry=node_points)
        nodes_gdf = nodes_gdf.set_index('id')

        edge_lines = []
        for n_fr, n_to in subgraph.edges():
            f = nodes_gdf.loc[n_fr].geometry
            t = nodes_gdf.loc[n_to].geometry
```

```
            edge_lookup = G.get_edge_data(n_fr, n_to)[0].get('geometry', ␣
↪LineString([f,t]))
            edge_lines.append(edge_lookup)

        n = nodes_gdf.buffer(node_buff).geometry
        e = gpd.GeoSeries(edge_lines).buffer(edge_buff).geometry
        all_gs = list(n) + list(e)
        new_iso = gpd.GeoSeries(all_gs).unary_union

        # try to fill in surrounded areas so shapes will appear solid and␣
↪blocks without white space inside them
        if infill:
            new_iso = Polygon(new_iso.exterior)
        isochrone_polys.append(new_iso)
    return isochrone_polys
```

I am going to use the function on G.

[13]:
```
isochrone_polys = make_iso_polys(G, edge_buff=25, node_buff=0, infill=True)
```

I need to create an empty geopandas GeoDataFrame.

[14]:
```
better_isos = gpd.GeoDataFrame()
better_isos['geometry'] = None
```

Now I am going to loop through the polygons and put them in a geodataframe.

[15]:
```
for i in range(len(isochrone_polys)):
    better_isos.loc[i,'geometry'] = isochrone_polys[i]
    better_isos.loc[i,'time'] =  str(trip_times[i]) + ' mins'
better_isos
```

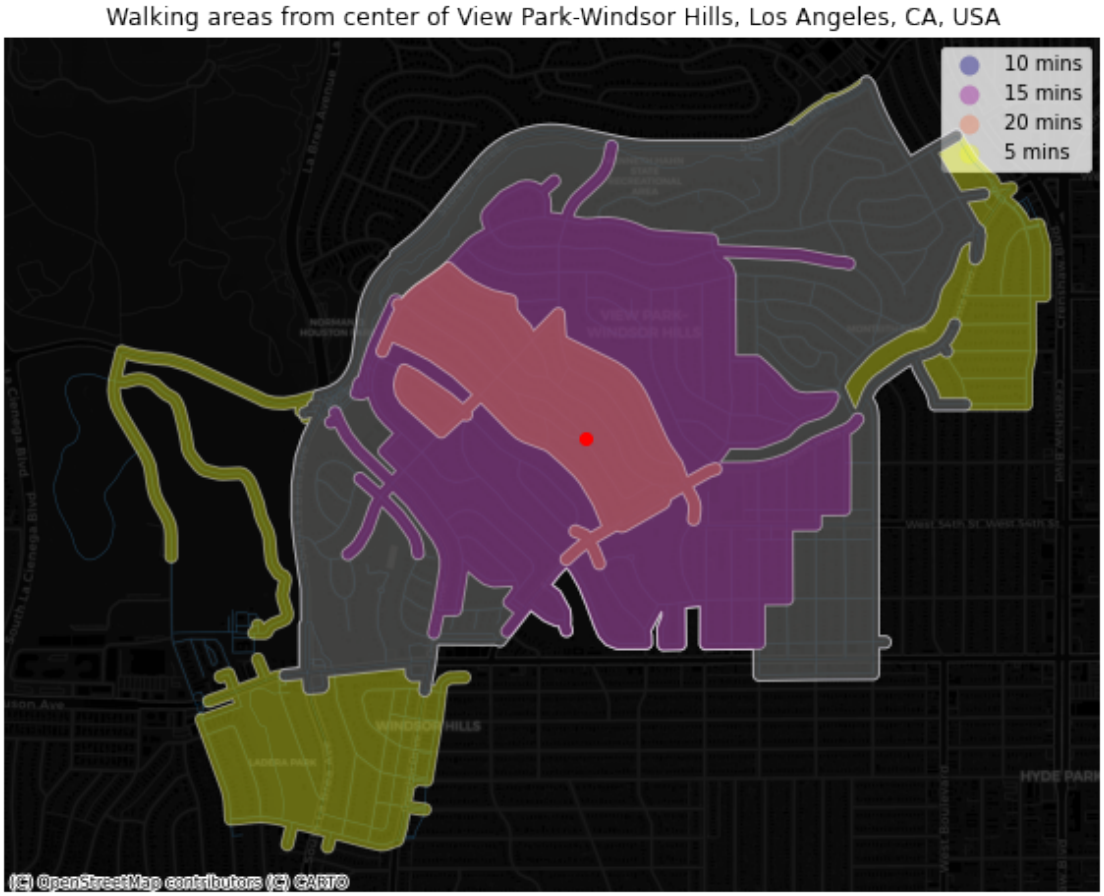
[15]:

	geometry	time
0	POLYGON ((-13175342.298 4026542.471, -13175343…	5 mins
1	POLYGON ((-13175103.404 4027231.158, -13175111…	10 mins
2	POLYGON ((-13174159.470 4027354.837, -13174160…	15 mins
3	POLYGON ((-13174465.071 4027750.468, -13174473…	20 mins

Now I should be able to create my isochrone map.

[16]:
```
# create a beautiful map with all relevant layers
# set up the subplots
fig, ax = plt.subplots(figsize=(10,15))

# add the isochrones
better_isos.plot(alpha=0.4,
                 ax=ax,
                 column='time',
```

```
                cmap='plasma',
                edgecolor='white',
                 legend=True,
                zorder=20)

# add the center node in red
gdf_nodes.loc[[center_node]].plot(ax=ax,color='r', zorder=30)

# add all nodes
# gdf_nodes.plot(ax=ax,
#                 markersize=1,
#                 zorder=10)

# add the edges
gdf_edges.plot(ax=ax,
               linewidth=0.5,
               alpha=0.2,
               zorder=10)

# hide the axis
ax.axis('off')

# give it a title
ax.set_title('Walking areas from center of ' + place)

# add the basemap
ctx.add_basemap(ax,source=ctx.providers.CartoDB.DarkMatter)
```



1.5 Conclusion

It seems it all worked correctly except for the fact that the color labeling doesn't make sense. The 5 minutes is shown to be the farthest instead of the closest. Also, the key is out of order. I couldn't figure out how to fix it.